Single-cycle Processors

Author	 Lee Edith.D
Last Update	@March 1, 2022

What's in a Processor?

Generic implementation

- Instruction Fetch
  - `op | rs | rt | rd | shamt | funct` = `MEM[PC]`
  - R(rd) ← R[rs] + R[rt] , take *ADDU* as an example
  - PC ← PC + 4, except *Branch*
- Decode
- Execute
- Memory
- Register Write

How to Design a Processor

- Analyze instruction set using RTL
- Select set of datapath components and establish clocking methodology
- Assemble datapath meeting the requirements
- Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- Assemble the control logic

Step1: Requirements of the Instruction Set

- MEM
  - inst. & data
- Register
  - Read *rs*
  - Read *rt*
  - Write *rt* or *rd*
- PC
- Extender (sign / zero extend)
- Add/Sub/OR unit for operation on register or extended immediate
- Add 4 (+ maybe extend immediate) to PC

Step2: Components of the Datapath

- Combination Elements
  - Adder
  - MUX
  - ALU
- Storage Elements
  - Register File
    - outputs: **busA, busB**

        - input: **busW**
        - register selector: **RW** (select the register to be written via busW)
        - output selector: **RA, RB** (select the register to be put on busA or busB**)**
    - Idealized Memory
        - input: **Data_in**
        - output: **Data_out**
        - **Address** selects the memory work to be written via the Data in Bus
- Clocking Methodologies

> define when data in a state element is valid and stable relative to the clock

    - Typical execution
        - read contents of state elements → send values through combination logic → write results to one or more state elements
        - **Cycle Time = CLK-tp-Q + Longest Delay Path + Setup + Clock Skew**
        - **Write** occurs only when both the writer control is asserted and the clock edge occurs
        - All storage elements are clocked by the same clock edge

Step3: Assemble Datapath meeting our Requirements

Fetching Instructions

- Read the inst. from Instruction Memory: **Instruction=M[PC]**
- Update the PC value to be the next address of the next inst. **PC ← PC+4**

Decoding

- Send the inst. fetched's **OP(6)** and **FUNCT(6)** to the **control unit**
- Read 2 values from the register file

Executing

- R-type
  - `R[rd] ← R[rs] op R[rt]`
  - OP and FUNC decodes ALUctr, specify manipulation in ALU
  - RegWr is the enable signal
  - rs, rt select the corresponding busA, busB to ALU, waiting to calculate
  - Result from ALU calculation will be put on busW to update register file, rd determines the object address
  - Waiting CLK and RegWr and update
- I-type (OR Immediate)
  - `R[rt] ← R[rs] or ZeroExt(imm16)`
  - Rb: rt, don't care
  - Rw: mux(rd, **rt**), controlled by RegDst
  - mux(**ZeroExt(imm16)**, busB) and **busA** to ALU, controlled by ALUSrc
  - ALUctr: **or**
  - Dtapath
  - `sw rt, rs, imm16`
  - `beq rs, rt, L`
- I-type (Load/Store)
  - `R[rt] ← M[R[rs] + SignExt[imm16]]`
    - Addr ← R[rs] + SignExt(imm16)
    - R[rt] ← M[Addr]
    - PC ← PC + 4
  - `M[R[rs] + SignExt[imm16]← R[rt]`
    - Addr ← R[rs] + SignExt(imm16)
    - M[Addr] ← R[rt]

        - PC ← PC +4
- I-type (Branch)
    - Condition ← R[rs] - R[rt]
    - if (true): PC←PC+4+(SignExt(imm16)<<2)
    - if (false): PC←PC+4
- J-type (Jump)
    - PC[27:0] ← M[PC] [25:0] <<2

Step4: Adding Control

Assemble Control Logic

- main control

> generate control signals except ALUctr

    - input
        - op (6)
    - output
        - ALUop (3)
    - PLA
- ALU control

> local decoding, generate ALUctr

    - input
        - ALUop (3)
        - func (6)
    - output
        - ALUctr (3)